Metrics Correlation and Analysis Service (MCAS)

Andrew Baranovski [*], Dave Dykstra, Gabriele Garzoglio, Ted Hesselroth, Parag Mhashilkar, Tanya Levshina

Fermi National Accelerator Laboratory, Batavia, IL, USA

{abaranov, dwd, garzoglio, tdh, parag, tlevshin}@fnal.gov

Abstract. The complexity of Grid workflow activities and their associated software stacks inevitably involves multiple organizations, ownership, and deployment domains. In this setting, important and common tasks such as the correlation and display of metrics and debugging information (fundamental ingredients of troubleshooting) are challenged by the informational entropy inherent to independently maintained and operated software components. Because such an information pool is disorganized, it is a difficult environment for business intelligence analysis i.e. troubleshooting, incident investigation, and trend spotting. The mission of the MCAS project is to deliver a software solution to help with adaptation, retrieval, correlation, and display of workflow-driven data and of type-agnostic events, generated by loosely coupled or fully decoupled middleware.

1. Introduction

The Grid is a common paradigm for sharing distributed resources. With the paradigm now arguably surpassing one decade of adoption by a wide variety of communities, the amount of information provided by distributed services, such as monitoring, discovery, troubleshooting, accounting, auditing, etc. is becoming increasingly difficult to manage in a coherent fashion. Because of the disjoint nature of all these data sources, the aggregation, transformation, and display of this distributed information is particularly challenging. To address this problem, the Fermi National Accelerator Laboratory has initiated the Metrics Correlation and Analysis Service (MCAS) project [1].

A core idea of this project is to factor out presentation and business analysis logic from available monitoring solutions into a standalone model, supporting common standards in data manipulation and presentation. The MCAS project prototyped several services, which rely on common techniques for data and information display aggregation. In particular, we've chosen portlet [2] technology to compose troubleshooting and metrics analysis dashboard, and XAware [3] to drive the data-integration model.

Section 2 introduces a problem in analyzing disjoint data sources and proposes data co-display and integration as possible solutions. These solutions are discussed further in the following two sections: section 3 discusses advantages and disadvantages of available technologies for data co-display; section 4 describes the problems in data integration. Section 5 delves into the description of the MCAS

[*] To whom any correspondence should be addressed.

architecture. Sections 6 and 7 discuss respectively the current and future work on MCAS. We conclude with section 8.

2. Uniform analysis of uniform data

Typically, operational problems cannot be easily visualized using any one particular display method. As a part of the decision-making process, the experts employ a variety of tools, which report data patterns that pinpoint incidents of already known problems.

This variety is built on technologies that often incorporate incompatible data input and produce output that cannot be used within the same context. These incompatibilities restrict the ability to perform analysis across data domains. A solution to overcoming these limitations consists in promoting intuitive ways to process and represent underlying monitoring or diagnostic information. In short, this solution promotes a uniform mechanism for analyzing uniformed data.

Phase 1 of the MCAS project has focused on a minimalist approach to describe data in XML and to build a foundation to support complex metrics analysis and presentation. In this phase, the project has focused on creating a toolkit of display and analysis tools for presentation and co-display of data. The idea behind the toolkit is to incorporate typical troubleshooting practices into components that can be hosted by a common container environment. The design and implementation of the delivered software are based on concepts of information display and data integration.

3. Data co-display

A data co-display is one of the simplest ways to cross-analyze the data. In the MCAS project, data co-display is implemented by aggregating independently designed and managed frames in a single web browser window. Each frame renders a different aspect of the input data. Not only does this approach decouple development and testing of the display components, but it also allows users to choose and change the presentation layout along with configuration details of each component, in a way that best represents the state of the system. As such, we have evaluated two technologies for content aggregation: web widgets and JSR 168 Java Portlets.

Both web widgets and JSR 168 Java Portlets offer a framework for developing code that can coexists in the context of a single web page. Ultimately, such framework allows developers to plug in the codes into service containers (iGoogle, Netvibe , MySpace). The function of the container is to augment all code with features that are common across all elements of the display. These features include widgets setup and display position, configuration editing, help, and persistence of configuration. Web widget containers generate a front page using the developer's scripts together with the framework code required to support the common functions. Finally, the generated code is rendered by the browser.

The web widget approach is extremely scalable, as it is entirely client-based and does not require a server side context. The significant drawback of this technology is in its dependence on the provider's API, resulting in vendor lock-in for hosting and support. This problem, however, can be mitigated by thin sub-framework wrappers, which isolate project-wide widget behavior.

In contrast to web widgets, JSR 168 Java Portlet specification is entirely server-side based. The specification sets forth a standard of development for java servlets, which generate independent, dynamic content. This context can be handled by the hosting environment for common management, configuration, and look and feel policies. Because the portlet specification is server-based, its major deficiency is scalability and the inherent complexity of the code. We believe that these weaknesses, however, are tolerable compared to the vendor lock-in, which is the principal disadvantage of web widgets. In addition, the specification clearly benefits from an open standard and a variety of implementations, available as community driven products (JBoss[4], Pluto[5], eXo[6]). In the MCAS project we have chosen to use java portlet specifications implemented by the JBoss portal as the primary means of composing and rendering project's user interfaces.

4. Data integration

Our goal is to provide unified data analysis capabilities on disjoint, format-incompatible datasets. We approach this problem through the unification of the data itself. This unification can be accomplished via data format and semantic transformation. These transformations make the data compatible with display and analysis tools, enabling the analysis of the unified data. Uniformity of format and common semantics decouples the design of the analysis software from the details of original input data. This also allows the reuse of the transformed data in contexts not originally envisioned by the project, a major benefit in enabling the utility of the solution for perhaps several years. In general, the drawback of this approach is the high startup cost of developing transformations of several data formats, while providing a usable model for describing the transformed data. In summary, the data integration layer must be able to interface to systems that function with diverse data sets.

The question remains as to what format the unification (or data description language) must adhere to. In the MCAS project we have decided against attempting to set a specific format that fits all possible cases, rather we have chosen to follow a solution that can capture common traits without unreasonably limiting future choices.

In this context we have isolated two requirements: support of structured data and support for navigation of structured data. For the MCAS project, the first requirement is important for preserving the semantics of input. The second requirement is critical for enabling a formalism of transformation of that input. Among a variety of choices and available third party implementations (key-value pair, CORBA, ad hoc binary, XML, etc.), XML fits the task the best. The XML has naturally become a backbone supporting data unification as well as the only means of describing the transformation workflows necessary for that unification.

5. Architecture

The MCAS workflow is organized among four players – presentation layer, content management system, data integration layer, and data sources.

The presentation layer for the MCAS system is a portal or dashboard. This page has a unique address and typically offers the user a default view of the system. The content of the page is supported by the Content Management System (CMS). The CMS is responsible for composing independent interface elements called “Portlets” using

1) The address of the user request
2) Static configuration parameters of the individual portlet
3) Run time information provided by the data integration layer

5.1. The Content Management System

The Content Management System (CMS) renders each portlet (P1, P2, P3, and P4 as shown in the figure **1**) independently. Each portlet can be autonomously designed with some unique perspective of a particular system aspect. A collection of different portlets, describing different perspective of the system, can be put together to form a dashboard. The dashboard view reflects the state of the entire system at a given time. The MCAS toolkit is based on high level data presentation routines from Google, RRD [9] and lower level JavaScript code developed in-house. The toolkit accepts data represented in standard XML schema as its input. Each portlet is rendered using information provided by the data integration layer. The MCSA project uses JBoss as Content Management System.

The data integration layer accesses a set of data sources and uses a collection of rules to transform and aggregate the retrieved content. This layer is responsible for providing content display to the CMS. The output is returned synchronously to the requesting party, which may be a portlet or another data integration rule set. Content display is expected to be available “immediately” following user’s request. Hence, the data integration layer URL must have access to the data in real-time.

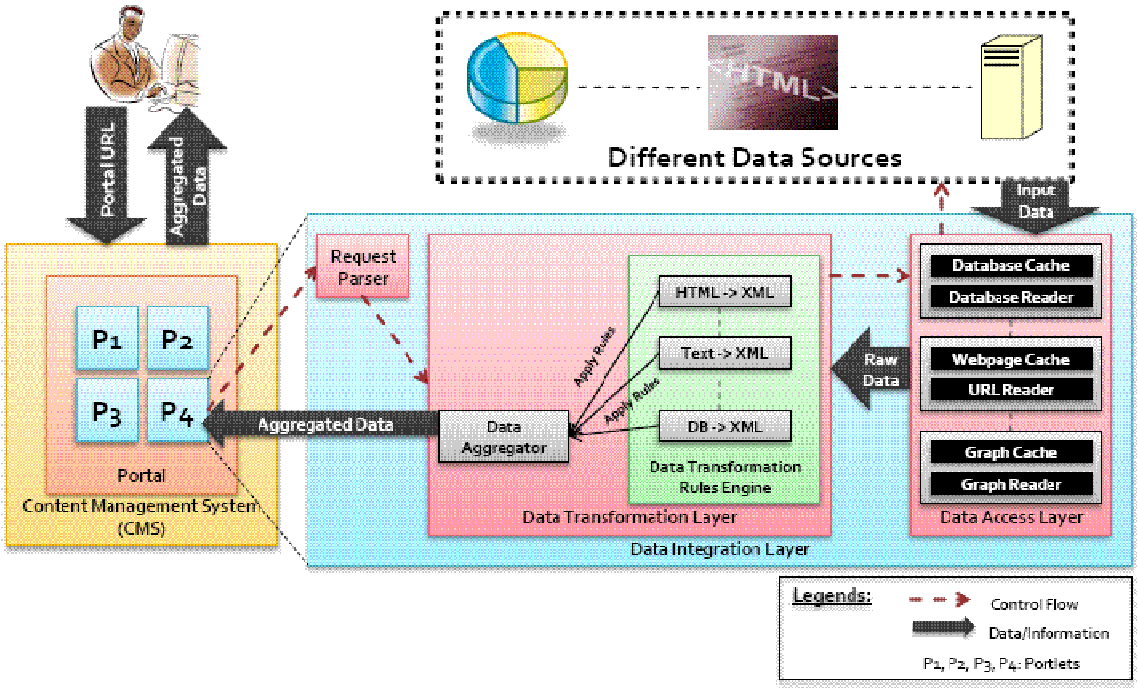

Figure 1: MCAS Architecture

In our model, the data integration layer defines all endpoints providing content to be displayed. These endpoints may be a cache of existing web pages or may invoke complex rules for transforming and aggregating data from other sources. The purpose of this layer is to respond to requests for data that is temporarily unavailable or in formats not compatible to the user interface. In fact, the compatible formats will typically imply requirements to data format, data quality, and data availability. Ultimately, the data integration layer will re-use existing preconfigured resources to generate the response. In doing so, it may need to restrict the quantity and format of the data retrieved from the endpoints, to generate a response within an acceptable amount of time.

A data source is a primary provider or the source of the pre-processed information. It may come in various formats and support variety of communication models. In the pilot stage of the project we focus on HTTP accessible resources.

5.2. Data integration model

The data integration model uses XAware for data source access, inter component data exchange and data transformation scheduling. XAware has extensive support for structured content based on XML and fairly rich set of features for content transformation. The product offers a well defined model supported by collection of Meta-XML documents. These documents define how the data sources should be combined to transform the original data into a more usable format. This model is expressed through the following three entities – a driver, a component, and a document.

An XAware driver is the meta-document which instructs how a data source can be accessed. It also specifies the data source address URL and other parameters required to retrieve content from the data source.

An XAware component is a logical view of the data source. It is a meta-document which defines the XML schema of the data available from the output and input drivers. Its purpose is to bind together the XML schema of the data source with the transformations of the data.

An XAware document is the ultimate product of the workflow. The document is built from the content provided by other documents or components. This model sets the structure of the output XML as well as defines the workflow of transformations for building that structure.

Each document name is mapped into URL endpoint accessible to the user through the XAware hosting environment. When user contacts such URL, the data integration layer triggers a series of transformations in strict accordance to the hierarchy of Meta object references comprising the overall document. The tree starts at the user contact point and ends with the driver. The resulting document is sent back in plain text using HTTP.

All Meta documents describing the workflow are part of the hierarchy of the resulting document. This design places limitations on flexibility of the output and options to dynamically influence the transformation sequence. In the initial phase of the project, these limitation are not critical. We have decided to invest into XAware features to transform informational content for presentation by the dashboard displays. However, given the weakness of the product, we are also investigating other platforms for data integration. One such alternative is Mule Enterprise Service Bus (ESB).

5.3. Mule Enterprise Service Bus (ESB)

We have evaluated Mule ESB [7] as the initial prototype suite for functions of the data integration layer. Attractive features of the Mule ESB are capabilities of accessing diverse data source transports, support for message based inter component exchange, concurrency, synchronization control and staged execution scheduling. One of the major features of Mule ESB integration platform is the ability to set up data and execution flow in a transport/interface agnostic way. In particular Mule ESB offers:

1) Codes to translate, or templatize translation of data formats
2) Options to manage synchronization with choices ranging from fully synchronous to Stage Event Driven Architecture (SEDA [8]) based solutions.
3) Codes which adapt out of the box to different transports (TCP, UDP, SMTP, FTP, jdbc, etc)

5.4. Messaging

Message exchange is a clever way to decouple the contexts of different programs. Messaging is a soft pattern which does not rely on a fixed db schema or file format. Rather, it is focused on data transport and synchronization issues. Consequently, within the Mule-message enabled infrastructure there are options for using opaque payloads; modeling of data access and aggregation is separated from specifics of the type-structure of the data sources. This is in contrast to the XAware model, in which transformation is assumed and all data exchanges have predetermined schema.

Messaging and data integration models have been used in an evaluation phase of the MCAS project to connect a set of message-enabled services into a workflow that refactors existing information portals. Information from a set of site efficiency sensors accessible through http for the D0 experiment at Fermilab [10] has been assembled in a Content Management System. Figure 2 below depicts the data and execution flow of the implementation, which uses formal data transformations and RRD processing engine to perform splitting, rescaling, and redrawing of the site efficiency data for the D0 experiment.

In our Mule workflow implementation we used a SEDA based message communication model. Mule messages communicate information and trigger component invocations. The generic Mule message consists of message envelope and enclosed opaque payload.

A necessary condition for supporting opaque content of the transport message is the availability of an interface that allows adapting that content to a format supported by the Mule endpoint components – a business logic container. This functionality is a major source of flexibility that allows developers to override Mule message end point interfaces and deploy customized transformations. Examples of such end point implementations are databases, mail servers, or RRD tool rendering engines. For the prototype evaluation phase we implemented an endpoint based on the RRD tool rendering engine. The interface to this RRD tool component allows a user to formulate simple data manipulations and set the

parameters of the resulting display. This module converts input data in the form of XML into a collection of RRD databases. The content of these databases is then used by the RRD tool to perform data manipulations using a command invoked from a template language similar to the example below.

```
ds(D0ProductionEfficiency)
ec=eff_code; ef=eff_fini;
RRD(CDEF:ec_adj=ec,0,100,
LIMIT CDEF:ef_adj=ef,0,100,
LIMIT LINE2:ec_adj#FF0000:eff_code(x100)
LINE2:ef_adj#0000FF:eff_fini(x100))
imgsize(600,300)
```

This meta-code directs the workflow to access D0ProductionEfficiency time series data source, split the data source content into two streams using values of data property ("eff_code", "eff_fini") and finally instantiates actual RRD command while referencing data factorized at the previous step.

The data transformation engine is built by setting up a model to describe message-driven interactions between Mule ESB message endpoints. This particular schema is designed to execute a template-like language and has only one data source (the production efficiency endpoint). The embedded RRD template enables transformations over split data streams. The result of the transformation is a new document with an image, which is sent to a portlet instance specifically configured to interact with this data integration model.

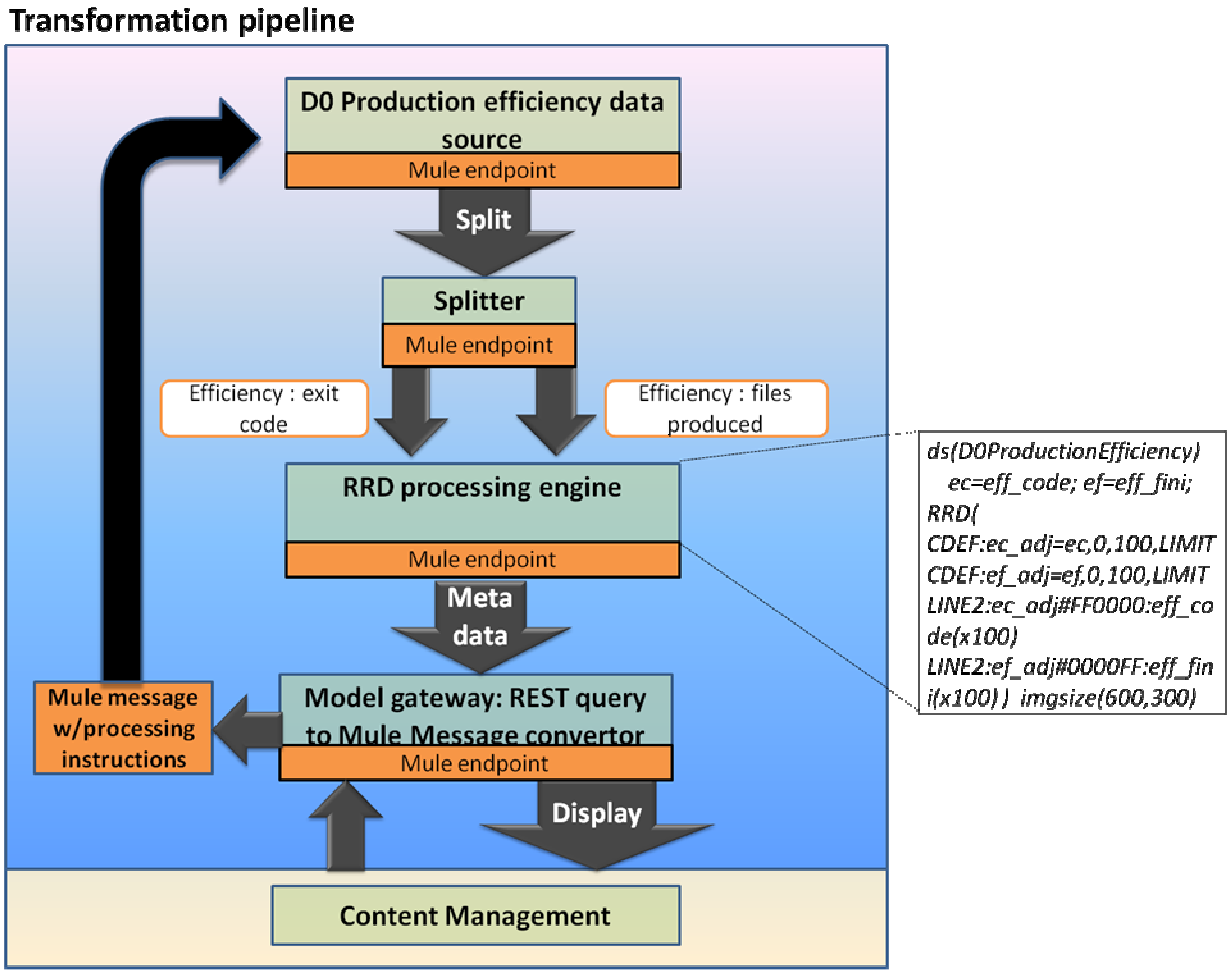

Figure 2 An example of data integration workflow

5.5. Architecture Summary

The Data Integration layer contains definitions of all URL endpoints that provide content for the display. These endpoints may be proxies to existing web pages or they may invoke rules for transforming and aggregating data retrieved from other sources. In all cases the data results in an XML representation of the resources accessed. The output of the proxies or rules is data of the semantics and format required by the portal Content Management System, which in this case is the JBoss portlet capability. The user interface is invoked in a web browser, displaying the portlets which have been produced from the aggregated data.

Though XAware as the data integration layer is strong in transformational capabilities, the static nature of its transformation sequence has led us to evaluate Mule as the integration component. Mule's focus on transport and synchronization issues allows separation of these factors from schema and format. We implemented a workflow in Mule to collect D0 production efficiency data, transforming it into images to be displayed via portlets.

6. Current work

At this stage of the project, the MCAS team focus is not on providing sophisticated means for analysis of distributed metrics or diagnostics data. Instead, the project is focused on development of easy to use and intuitive tools for displaying data already reported by existing information portals. Following that path, we have already developed a substantial collection of tools that fit the monitoring needs of Fermilab's USCMS facility operations. This software enhances the capabilities of existing experiments' portals by bringing the expertise we have gained from analyzing, monitoring, and troubleshooting use cases of users and operators of largely distributed software systems from several different experiments. The current capabilities of the software are described in the sections that follow. Fig. 3 shows a screen shot of a dashboard composed by Bar graph and Image display widgets.

6.1. Table view

Table view renders a data table model. Each row of the table undergoes summarization which determines the relation between columns of the model data and columns of the data displayed on the web page. The table view portlet supports custom sorting features, table size constraints, and user-assigned color coding of different weight. The example of the input data format is given below:

```
<table xmlns="http://www.fnal.gov/docs/products/mcas/TableView">
 <column name="c1">
  <row>value1</row>
 </column>
 <column name="c2">
  <row>value2</row>
 </column>
</table>
```

6.2. Bar graph

Bar graph displays a collection of indicators that visualize value pairs in relative proportions. These collections are often used to show health or performance status of the system relative to predefined metrics. The idea behind these widgets is to display information collected from a variety of "status" pages into a single compact document.

6.3. Time series

Time series widgets adopt the Google visualization API [11] for annotated timelines and visualize XML documents that display data changing in time.

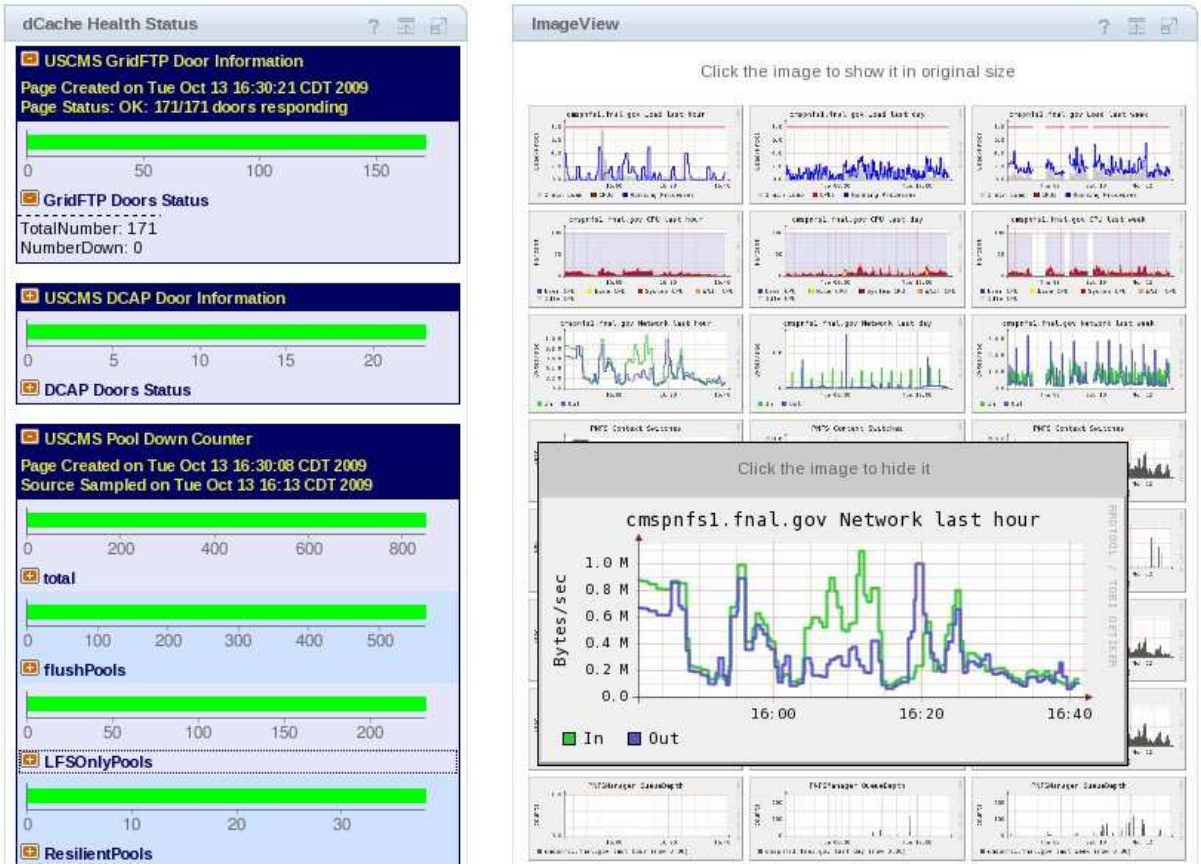

Figure 3 A screenshot of a dashboard that displays Bar graph and Image display widgets. The bigger plot at the bottom right is the zoom-in of one of the thumbnail-size images underneath.

6.4. Image display

The image display portlet accepts an XML document that encodes a list of images as URL locations. The portet generates HTML code that rescales those images such that they all fit inside a fixed-size portlet window. This portlet allows the user to select and combine image data generated by other services in order to increase the informational density of the dashboard page.

6.5. RRD data analysis and display tool

One of the goals of this project is to build a solution that offers a formal interface to the analysis of disjoint data sets. Perhaps the simplest approach to providing this feature is to re-use existing tools and their interfaces through a thin layer of templates and known agreement on how infrastructure should instantiate the actual interface invocation. Although we do not directly address the data analysis in phase 1 of the project, we prototyped an RRD driven service for template based transformation and display of XML time series.

7. Future work

One of the major obstacles to reusing the results of this work is the complexity of the underlying technologies. To overcome this problem, we now plan to focus on understanding common data integration patterns in order to isolate reusable service components. Ideally the roles of these components in a model may be changed by simple rearrangement of the workflow or reconfiguration of the parameters of the service.

8. Conclusions

In this project we have addressed the problem of building monitoring and analysis portals for systems that do not support common semantics for data describing system state. Our solution is based on technologies that allow data integration, transformation, and co-display. In the project we have focused on ease of refactoring and adaptation of data through a specially provisioned integration layer. We have used an XAware engine as the driver for that layer, leveraging its support for rules to implement unification of semantics and specifying the format of the input. For addressing the problem of heuristic correlation of information, we have leveraged a data co-display idea through building HEP experiments' dashboards using JBoss-backed portlet technology. In the future we'll focus on allowing more sophistication in how advanced data display tools can be used in conjunction with data generated by user-supplied transformation templates. The purpose of this work is to ensure a better fit of metric data analysis and correlation products to monitoring and troubleshooting requirements of users and facility operators.

Acknowledgments

Fermilab is operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy. This work was partially funded by the Center for Enabling Distributed Petascale Science (CEDPS) through the Scientific Discovery through Advanced Computing program (SciDAC2), Office of Science, U.S. Dept. of Energy.

References

[1] The Metrics Correlation and Analysis Service project:
http://www.fnal.gov/docs/products/mcas – Accessed on Oct 13, 2009
[2] The Java Community Process, JSR-000168 Portlet Specification
http://jcp.org/en/jsr/detail?id=168 – Accessed on Oct 13, 2009
[3] K Vandersluis 2004 XML-Based Integration with XAware: Unifying Applications and Data in Today's e-Business World *Gulf Breeze, FL: Maximum Press*
[4] M Fleury, F Reverbel 2003 The JBoss extensible server *Lecture notes in computer science* Springer
[5] The Apache Reference Implementation of the Java Portlet Specfication
http://portals.apache.org/pluto – Accessed on Oct 13, 2009
[6] eXo Platform – http://www.exoplatform.com/ – Accessed on Oct 13, 2009
[7] Mule Enterprise Service Bus – http://www.mulesource.org – Accessed on Oct 13, 2009
[8] Welsh M, Culler D, Brewer E 2001 SEDA: An Architecture for Well-Conditioned, Scalable Internet Services *Proc Eighteeth Symposium on Operating Systems Principles (SOSP18), Chateau Lake Louise, Canada*
[9] RRD Tools – http://oss.oetiker.ch/rrdtool – Accessed Oct 13, 2009
[10] The D0 Experiment – The D0 Collab., "The D0 Upgrade: The Detector and its Physics", Fermilab Pub-96/357-E.
[11] Google Visualization API – http://code.google.com/apis/visualization/ – Accessed Oct 13, 2009